Section 2: Week 4: Big Oh

Nate Bachmeier

TIM-8110: Programming Languages and Algorithms

May 26, 2019

North Central University

# Big Oh

An algorithm is a well-defined list of steps for performing an action. We use algorithms every day for an assortment of tasks, such as baking a cake. When you bake a cake, it can take anywhere from fifteen minutes to all afternoon. The quality of the produced result can also range from dried up bread to an award-winning masterpiece. What changes both the time and quality of the result is the implementation of the recipe (algorithm). Just as there are an infinite number of ways to implement a cake, there are infinitely large ways to implement computer algorithms.

## What is Big-O

The de facto method for measuring the efficiency of a computer algorithm is called Big-O notation. The notation then describes the maximum number of steps that are required to perform an action proportional to an input. For instance, an algorithm of O(n^2) complexity would operate on 10 items in at most 100 steps (10\*10). If an alternative implementation was O(log n) the same input would only be 4 steps.

Along with the worst-case scenario it is also useful to understand the (1) best case and (2) average case. Consider a simple algorithm to find a specific number within a set. The best case (omega function) occurs when the first item checked matches and there is 1 step required. The average case (theta function) will amortize to n/2 steps.

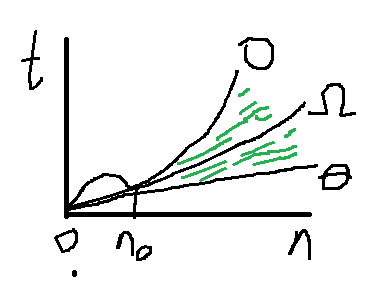
These three functions can then be used to describe the asymptotic range of how many steps are needed to perform the algorithm, proportional to a given input size. This information can empirically tell the user if the mechanism will work for their scenario. Perhaps the algorithm is inefficient, and this information simply drives the conversation around supported input limits. Even N! time is still manageable for small numbers of N.

# Challenges with Big-O

There are many challenges to using Big-O notation outside of the academic classroom. The first issue is that the length of a step is not guaranteed to be uniform across a domain, or even within the same problem. Consider the scenario where an array is traversed once, and each element passed to a transform function. The complexity of this this algorithm is equal to the length of the array O(n).

Inside of the transform different permutations are needed to handle different object types, resulting in entirely different code paths. Along these different paths there will cache hits and misses resulting in entire bodies of work that will be conditionally performed. Correctly accounting for these nuances requires a white box understanding of the entire implementation. For any reasonably complex production system this is difficult to even white board in its entirety.

Another challenge comes from the assumption that number of steps aligns with time to complete. If the contrived array transform function can be



**Figure 1: Big Oh, Theta, Gamma**